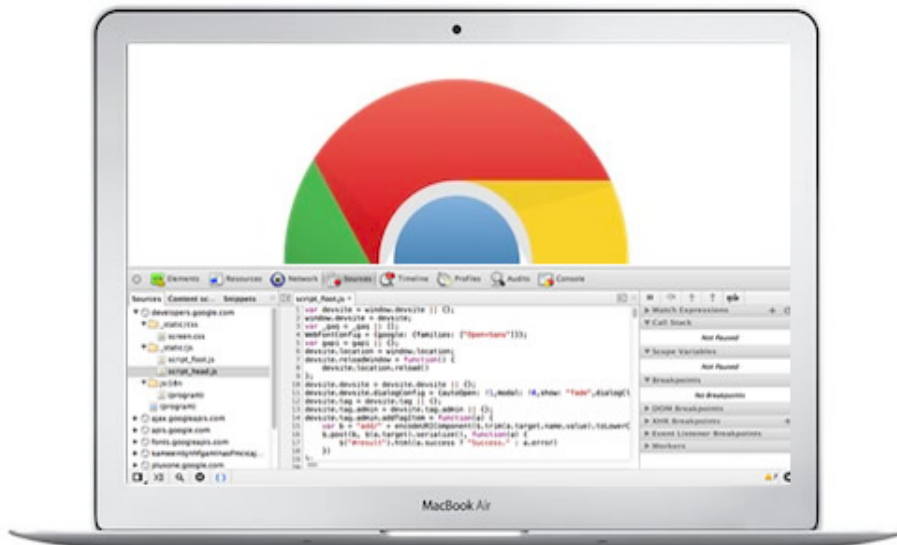


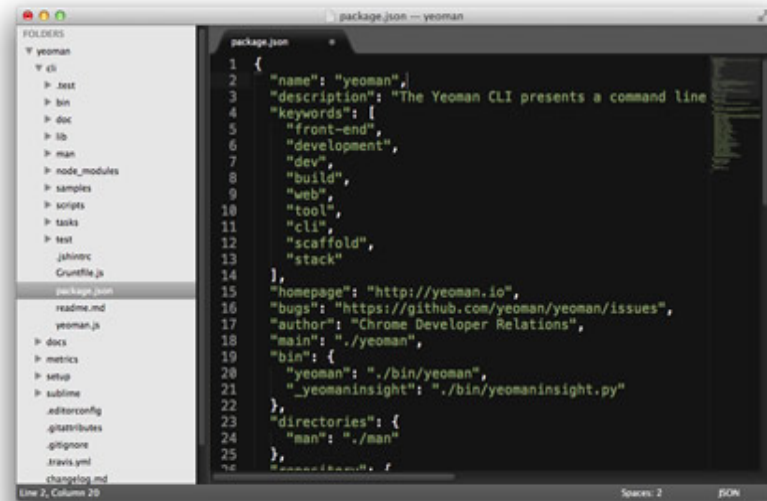


TOOLING  
FOR THE  
MODERN  
WEBAPP  
DEVELOPER

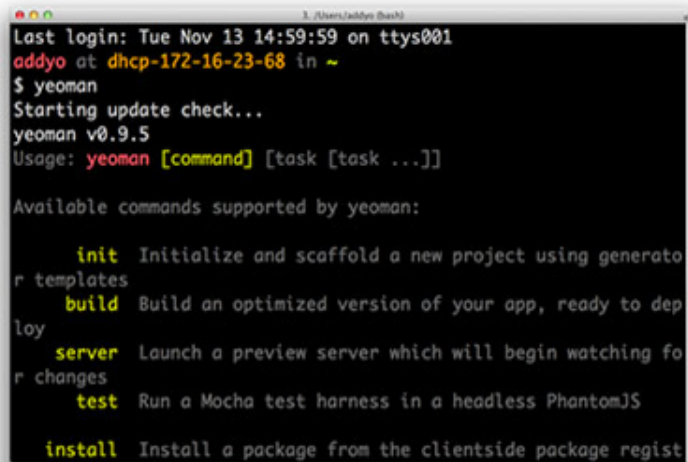
@addyosmani



Browser + DevTools



Text Editor



Terminal

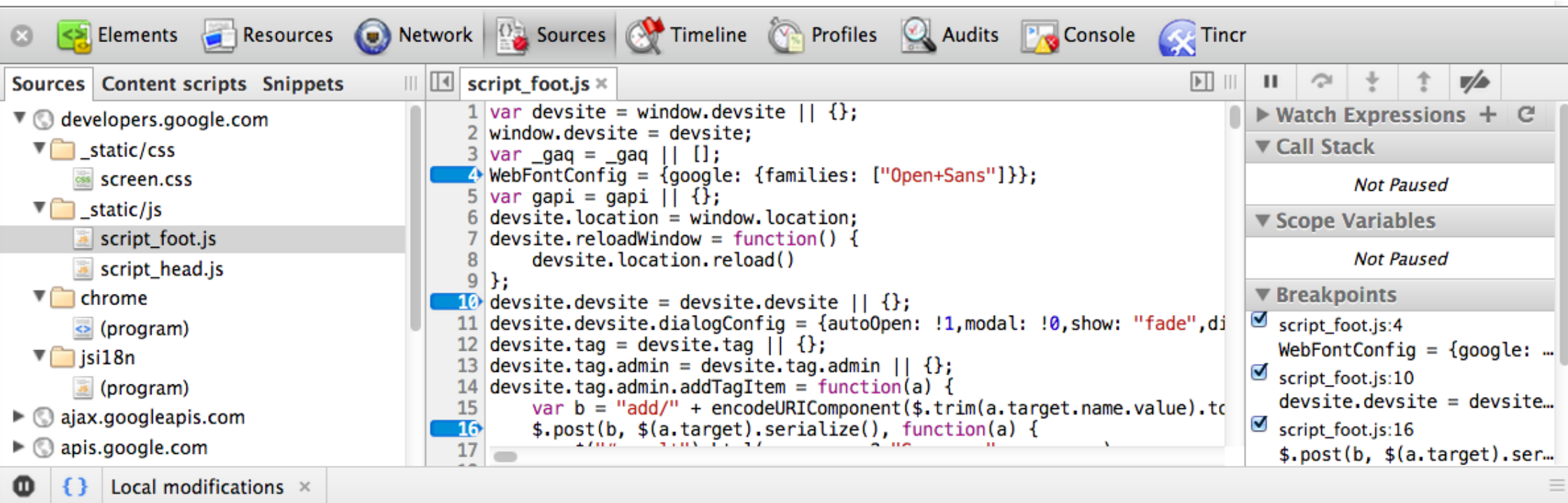


Devices



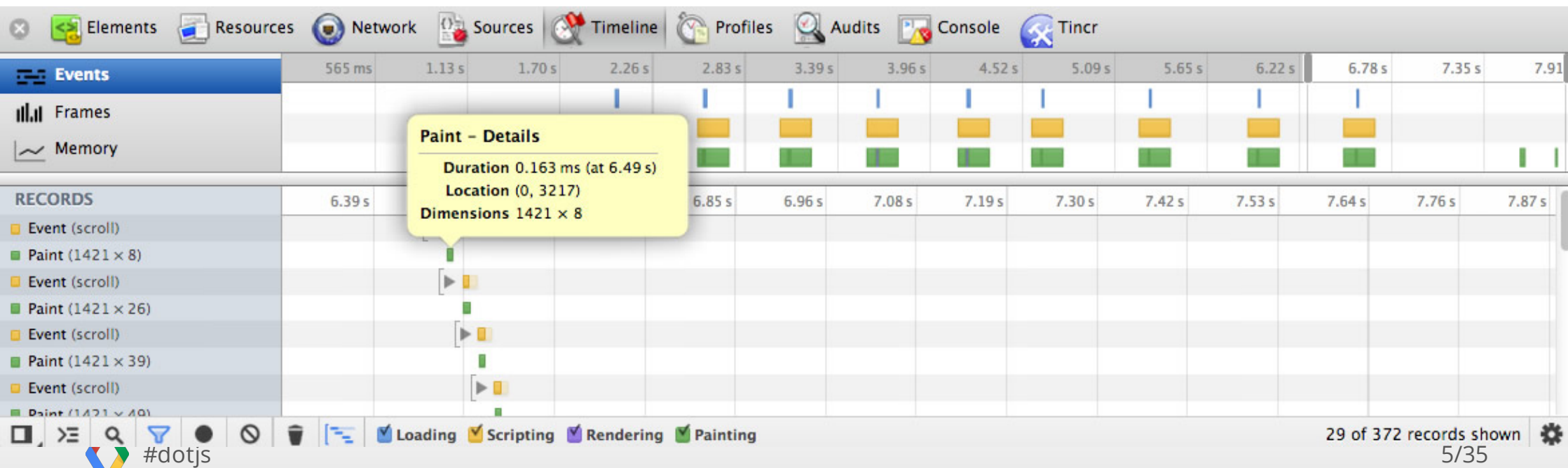
# In-browser DevTools

- Constantly evolving
- Use [Canary](#) channel for development
- Lots of juicy [experimental](#) features



# Performance: Timeline + Frames view

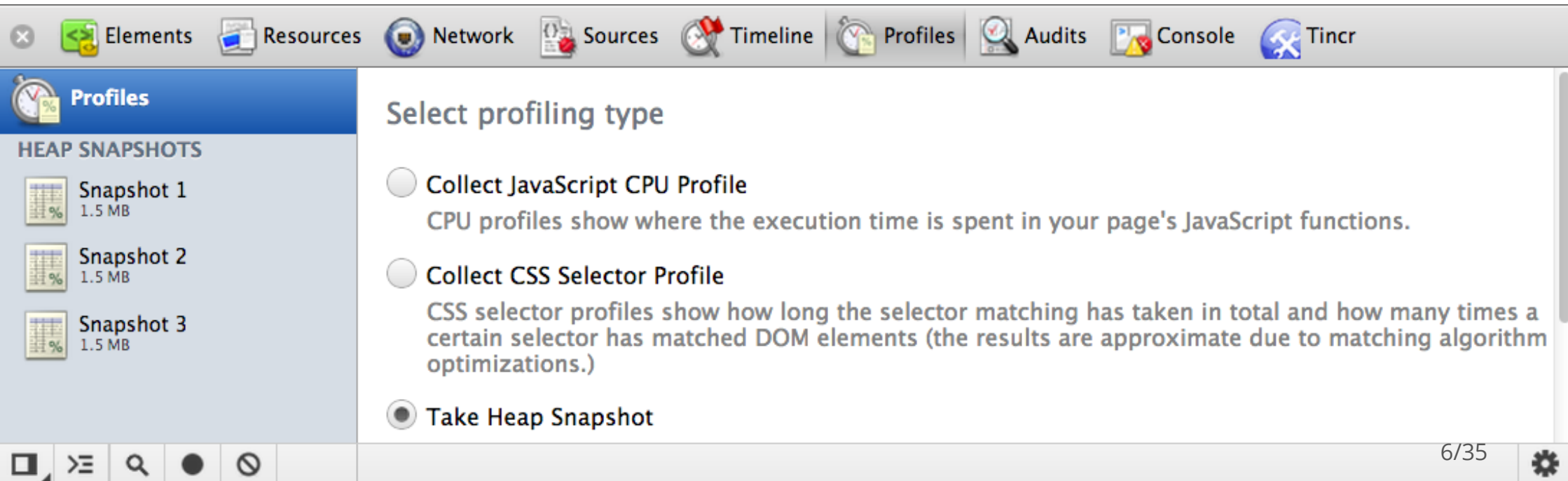
- Timeline gives you an overview of memory usage over time
- Summary and detailed views
- Helps remove jank. Layout or scripts - who triggered what?
- Frames view helps achieve that snappy 60fps you ideally want





# Finding memory leaks and DOM leaks

- JavaScript, CSS, Heap snapshot Profiles
- What is using memory at a given point in time? Not being GC'd?
- Use [comparison](#) view to identify potential memory leaks
- Use [summary](#) view to identify DOM leaks



The screenshot shows the Chrome DevTools interface with the Profiles panel open. The top toolbar includes icons for Elements, Resources, Network, Sources, Timeline, Profiles, Audits, Console, and Tincr. The Profiles panel is divided into two sections: 'HEAP SNAPSHOTS' and 'Select profiling type'.

**HEAP SNAPSHOTS**

- Snapshot 1  
1.5 MB
- Snapshot 2  
1.5 MB
- Snapshot 3  
1.5 MB

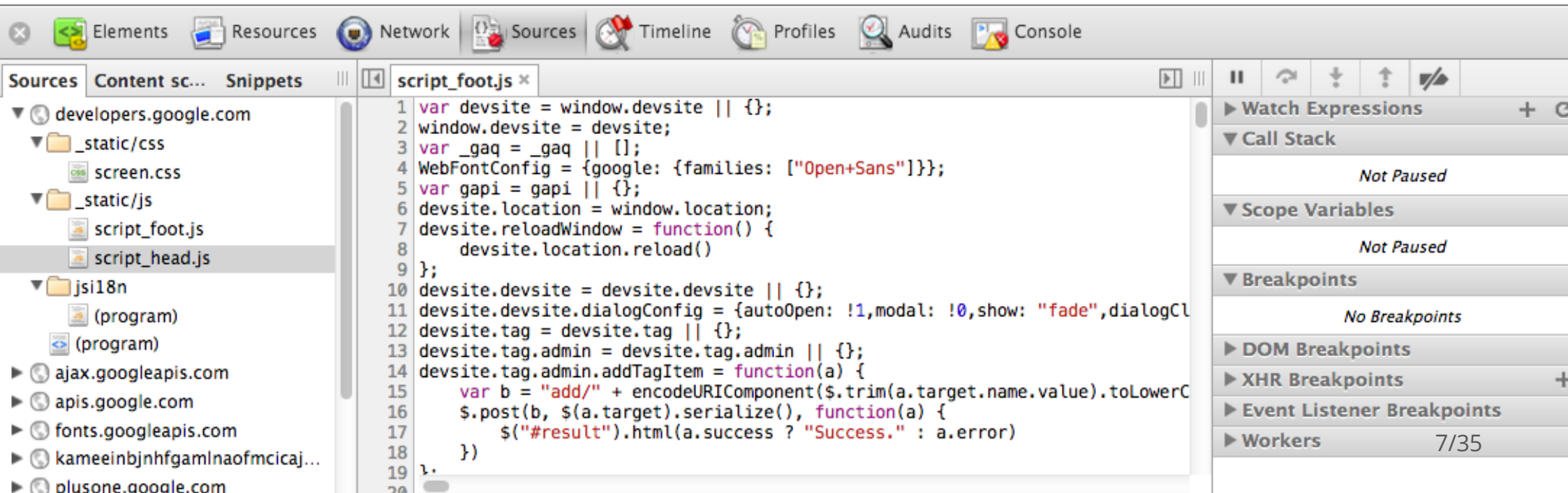
**Select profiling type**

- Collect JavaScript CPU Profile  
CPU profiles show where the execution time is spent in your page's JavaScript functions.
- Collect CSS Selector Profile  
CSS selector profiles show how long the selector matching has taken in total and how many times a certain selector has matched DOM elements (the results are approximate due to matching algorithm optimizations.)
- Take Heap Snapshot

At the bottom right of the interface, the page number '6/35' and a settings gear icon are visible.

# A better authoring workflow

- Live Edit
- [Snippets](#)
- Revision history
- [AutoSave](#)



# Live reloading + SASS Source Maps

- Editing compiled CSS has little value
- [Enable new SASS hotness](#)
- BOOM! You can edit SASS source files
- Changes automatically reload

## Settings

## Experiments

General

Experiments

Shortcuts

**WARNING:** These experiments could be dangerous and may require restart.

Snippets support

Native memory profiling

Live native memory chart

FileSystem inspection

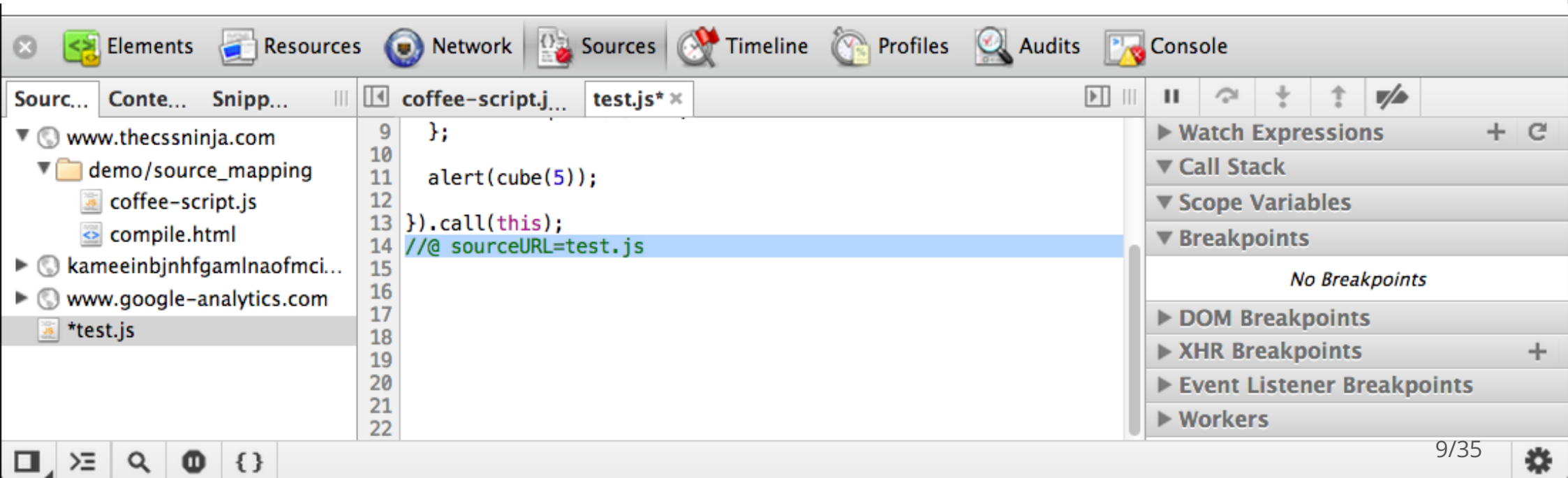
Canvas inspection

Support for SASS

Use CodeMirror editor

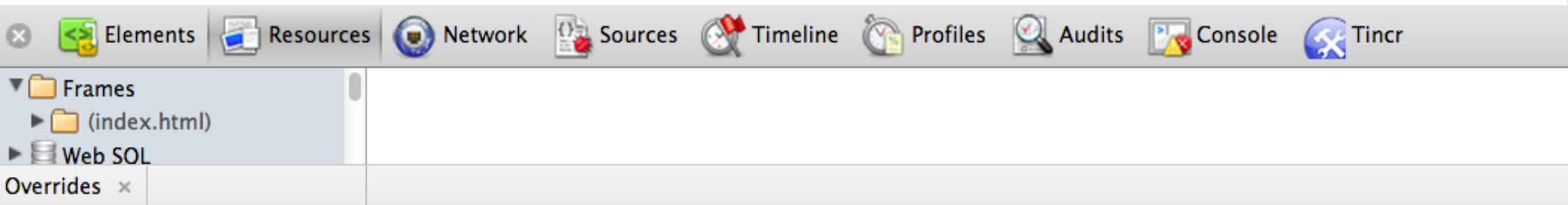
# CoffeeScript + [@sourceURL](#)

- Compile your Coffee sources
- Open up the DevTools
- Review your compiled file
- Whoa! [sourceURL](#) comments



# Mobile Debugging: Overrides Panel

- User Agent
- Device Metrics
- [Geolocation](#)
- [Orientation](#)
- Emulate touch events



## Overrides

User Agent

Internet Explorer 9

Override Geolocation

Geolocation Position: Lat =  , Lo

Emulate position unavailable

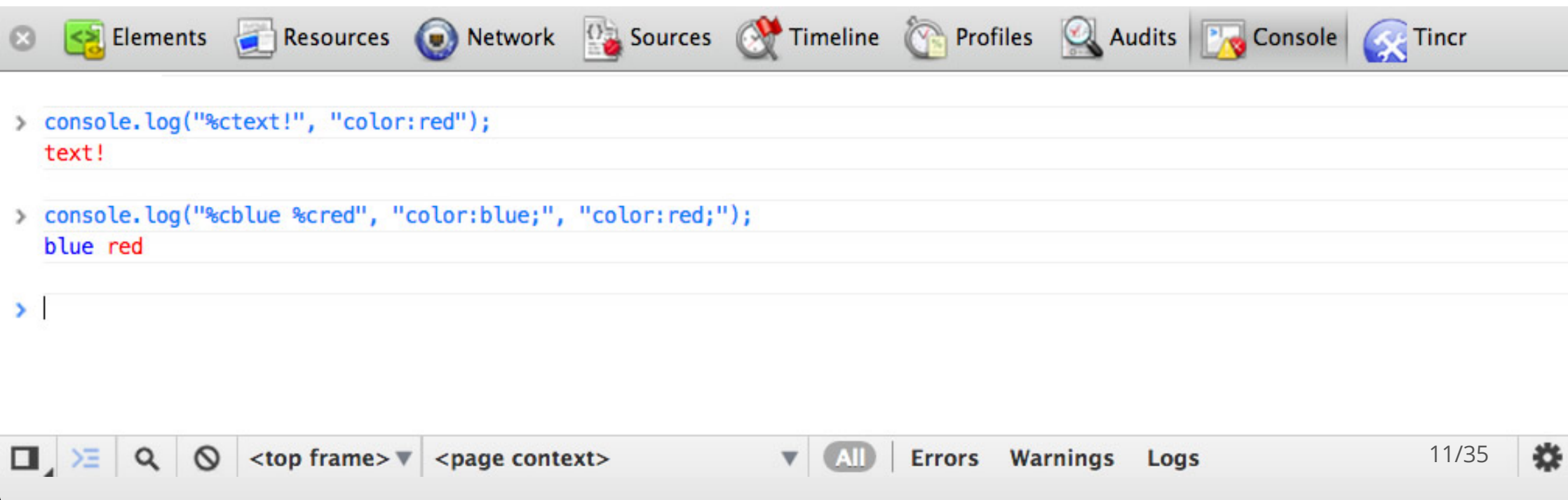
Device metrics

Override Device Orientation



# DevTools Console

- Styled-console
- Multi-style support
- inspect() command

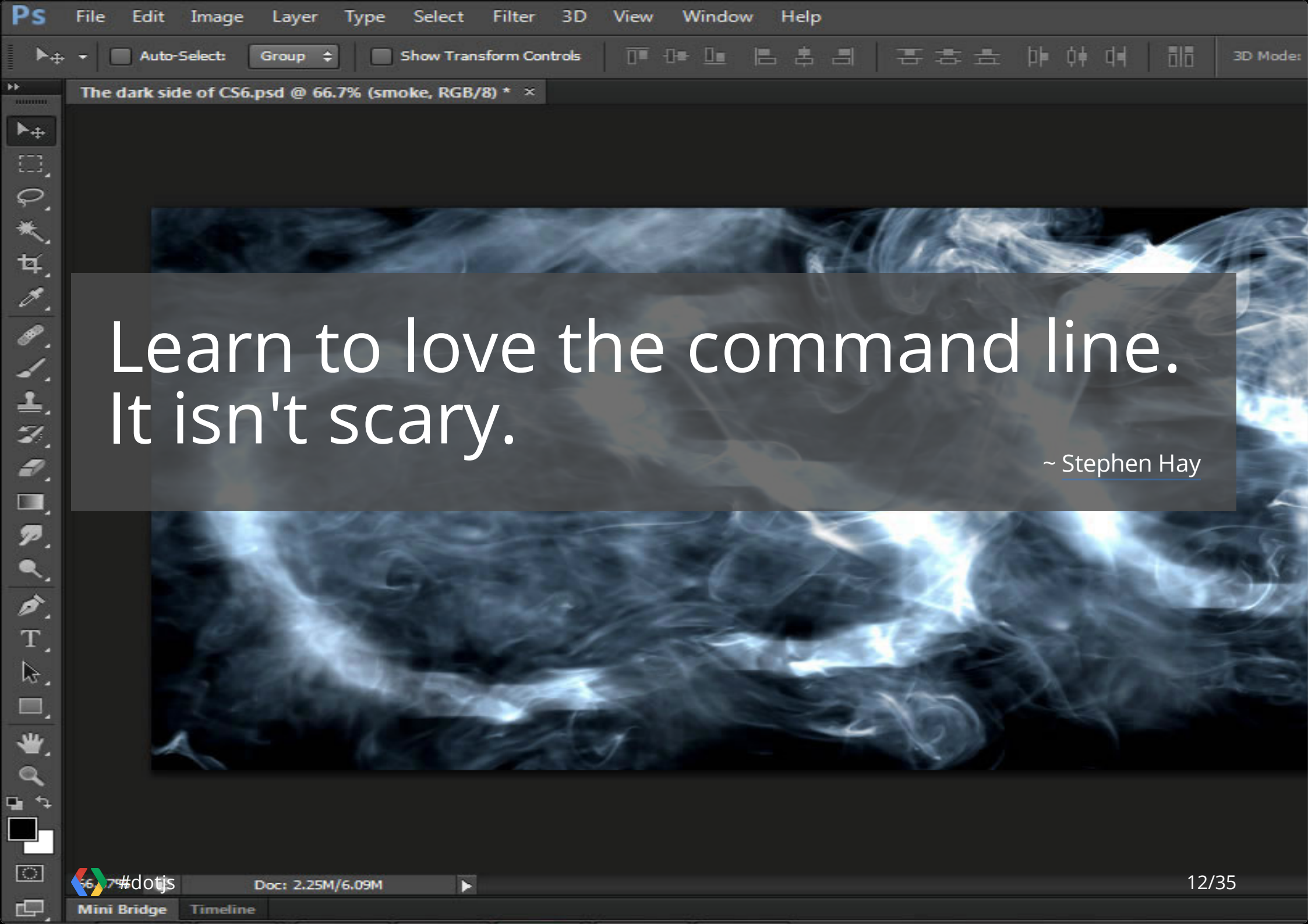


The screenshot shows the Chrome DevTools Console interface. The top toolbar includes icons for Elements, Resources, Network, Sources, Timeline, Profiles, Audits, Console, and Tincr. The Console panel displays two log entries:

```
> console.log("%ctext!", "color:red");  
text!
```

```
> console.log("%cblue %cred", "color:blue;", "color:red;");  
blue red
```

The bottom status bar shows the current context as <top frame> and <page context>, with filters for All, Errors, Warnings, and Logs. The page number 11/35 and a settings gear icon are also visible.



Learn to love the command line.  
It isn't scary.

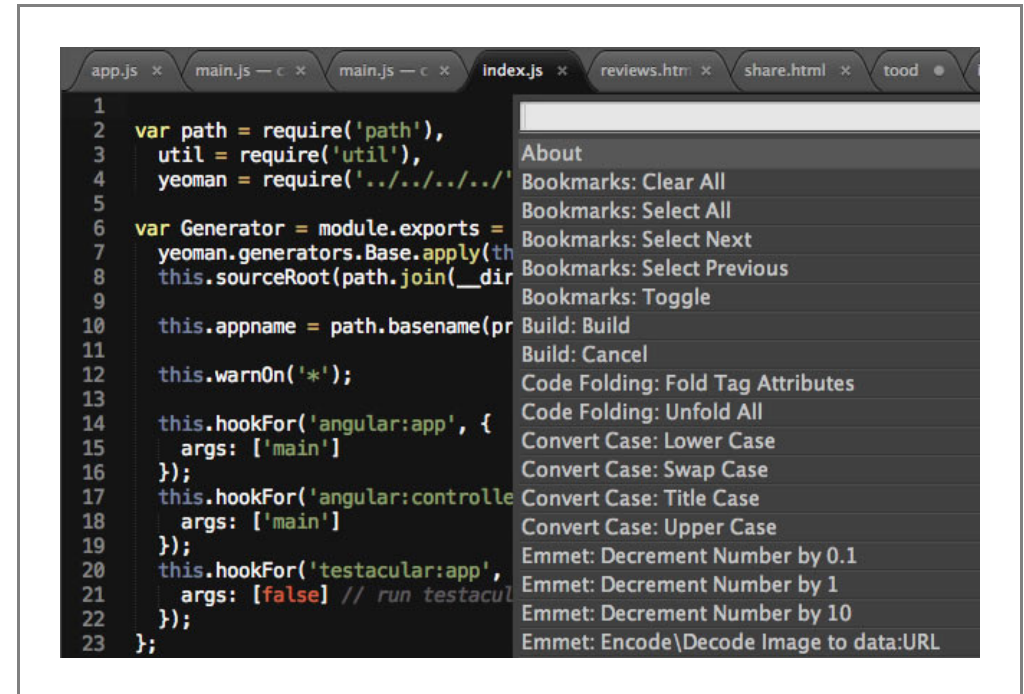
~ Stephen Hay

# Command-line

- Make it look [hot](#) ♡.
- Capture and [replay](#) your command line *history*
- [DOTFILE EVERYTHING](#) ([mine](#))
- [Aliases](#) are awesome
- Our faves: `gz` , `server` alias and [ny](#)
- Some for [Browserstack](#): `win7ie8`, `win8ie10`, `ios3`

# Sublime Text #protips

- Run JavaScript From Your Editor In The [Browser](#)
- Use the built-in [Build System!](#)
- Zen coding with [Emmet](#)
- Stack Overflow [code search](#)
- [Snippets](#) for frameworks





Finding missing semicolons should never be a manual process.

*write less, do more.*

THANKS OUR GENEROUS SPONSORS



ASP.NET



wijmo





# Linting

- On file save
- On source control commit
- At build time
- *Anything better?*

```
→ grunt
Running "lint:files" (lint) task
Lint free.

Running "qunit:files" (qunit) task
Testing jquery.demo.html...OK
>> 4 assertions passed (35ms)

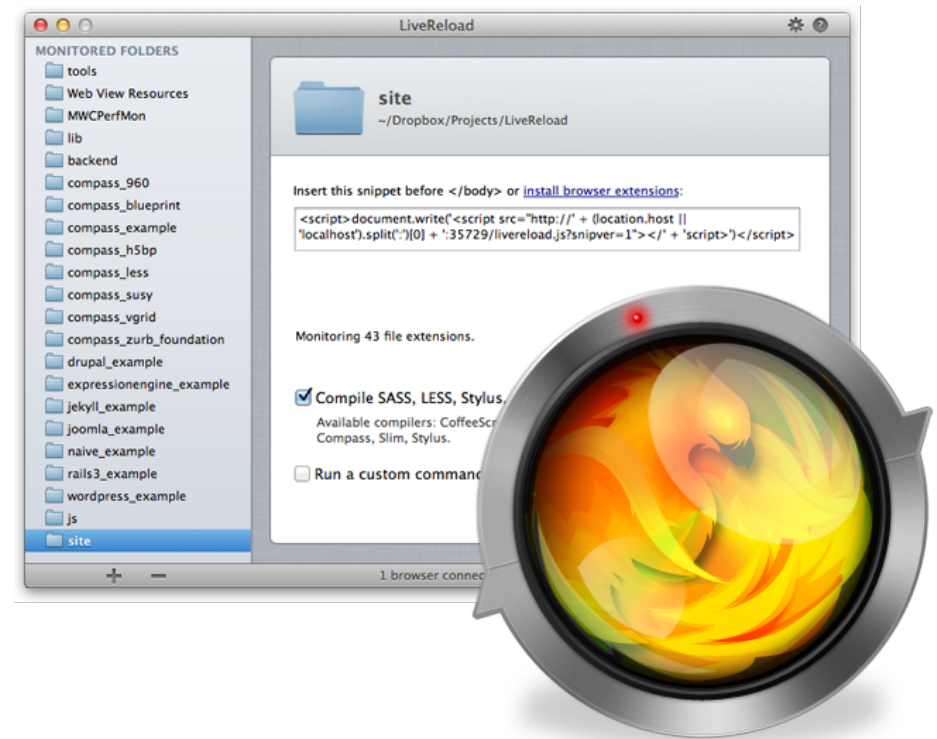
Running "concat:dist" (concat) task
File "dist/jquery.demo.js" created.

Running "min:dist" (min) task
File "dist/jquery.demo.min.js" created.
Uncompressed size: 455 bytes.
Compressed size: 225 bytes gzipped (312 bytes minified).

Done, without errors.
```

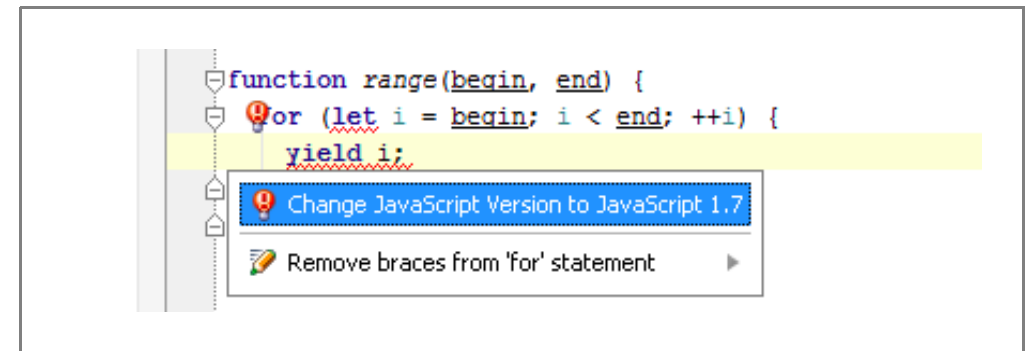
# Live feedback

- [Linting](#)
- [Reload](#)
- Recompilation



# WebStorm

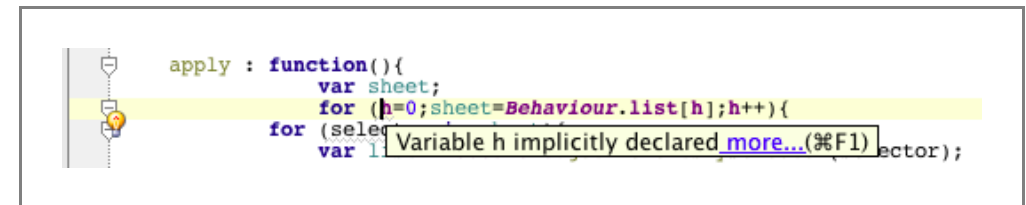
- [Live Edit + Chrome](#)
- JS Language version based suggestions
- Code inspection and zen coding
- Suggestions for DRYer code
- Built in code linting
- and [more](#)



```
function range(begin, end) {  
  for (let i = begin; i < end; ++i) {  
    yield i;  
  }  
}
```

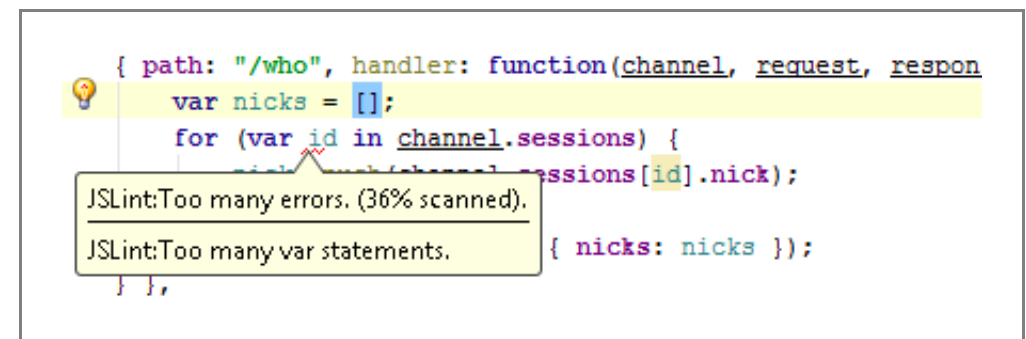
Change JavaScript Version to JavaScript 1.7

Remove braces from 'for' statement



```
apply : function(){  
  var sheet;  
  for (h=0;sheet=Behaviour.list[h];h++){  
    for (select  
    var i
```

Variable h implicitly declared more... (%F1)



```
{ path: "/who", handler: function(channel, request, respon  
  var nicks = [];  
  for (var id in channel.sessions) {  
    nicks.push(channel.sessions[id].nick);  
  }  
  { nicks: nicks }  
}
```

JSLint: Too many errors. (36% scanned).

JSLint: Too many var statements.

# Unit Testing In The Cloud

## Testing approaches you already know:

- In [the browser](#)
- In a headless browser on-demand via cmd line: `grunt qunit`
- In a headless browser [post-push](#)

## New hotness:

- In multiple browsers *in the cloud* via cmd line:

```
bunyip -f Modernizr/test/index.html -c ~/bunyip/config.js -b ios
```

```
bunyip -f index.html local -l "firefox | chrome | safari | phantomjs"
```



# Build system



Lint. Resolve dependencies. concatenate modules. compile. Flatten your CSS @imports. Remove debugging statements. Compress images. Precompile templates. Run tests in a variety of environments. Revs asset paths for caching. Affirm code quality.



# New hotness: Grunt.js



# GRU

Grunt is a task-  
JavaScript proje

[View on GitHub](#)

[Docu](#)

## Getting started

Be sure to read the getting started guide, which is a complete guide to configuring grunt for your project. In addition, check out the [example gruntfiles](#) which highlight a number of fairly common configurations.



Install using `npm install -g grunt`

## Latest pl

[contrib-ember](#) Manage and

[contrib-manifest](#) Precompile

[pistachio-compiler](#) Grunt task t

24/35

# Grunt.js:

- Task based command line build tool
- Alternative to Rake/Cake/Make/Jake
- Rich community of [build tasks](#)
- Generates simple skeleton for new projects
- Lint, test, concat, watch and min out of the box.
- *however..you're still responsible for workflow*

```
answering "?" to any
question will show question-specific help and answering "
none" to most questions
will leave its value blank.

Please answer the following:
[?] Project name (1) toolsRawk
[?] Description (The best project ever.)
[?] Version (0.1.0)
[?] Project git repository (git://github.com/addyo/toolsR
awk.git)
[?] Project homepage (https://github.com/addyo/toolsRawk)

[?] Project issues tracker (https://github.com/addyo/tool
sRawk/issues)
[?] Licenses (MIT) |
```

*so much choice! you want flexibility.  
how could we make this any easier?*

# Introducing Yeoman









Limit the time spent on writing boilerplate for your app





Automate as much of your workflow as possible





# Yeoman

- Authoring abstractions
- Scaffolds
- Linting
- LiveReload
- Testing
- Build tools
- and more.



Built on top of great tools like Grunt and Bower



TWITTER ENGINEERING PRESENTS

# BOWER

THE BROWSER PACKAGE MANAGER

HTML, CSS, AND JAVASCRIPT

[GITHUB](#)





# Walkthrough

1. Package management
2. Generators
3. Live Reload
4. Testing
5. Build system



# What's next?



# Yeoman in 2013

1. Improved flexibility with tools (Grunt, Bower)
2. Better editor integration
3. Support for backends (Rails, PHP)
4. Better mobile helpers and remote debugging
5. Support for all of JS.next (Traceur)
6. Deployment (Heroku, AppEngine)
7. ..plans for a little [more](#)



# Yeoman 0.9.5 Just Launched!

- Visit [yeoman.io](http://yeoman.io)

The screenshot shows the Yeoman website homepage. On the left is a navigation menu with links: Home, Why Yeoman?, Getting Started, Documentation (with sub-links for Command Line, Package Manager, Installation, and Insight), FAQs, Discuss, Google Plus, Twitter, GitHub, and Contributing. The main content area features the Yeoman logo (a cartoon character in a top hat) and the name 'YEOMAN'. Below this is the tagline 'MODERN WORKFLOWS FOR MODERN WEBAPPS' and a paragraph describing Yeoman as a robust set of tools for building web apps. A video player is embedded, titled 'Introduction to Yeoman', showing a terminal window with code and a speaker's video feed. The terminal output includes a welcome message and a question about including Twitter Bootstrap for Compass.



# Learn to love your workflow and tools



g+      addyosmani.com/+  
twitter    @addyosmani  
www      addyosmani.com  
github    github.com/addyosmani



# Thank you!